Know the Ropes: A Heuristic Strategy for LLM-based Multi-Agent System Design

Anonymous ACL submission

Abstract

Single-agent LLMs hit hard limits—finite context, role overload, brittle domain transfer. Conventional multi-agent fixes soften those edges yet expose fresh pains: ill-posed decompositions, fuzzy contracts, and verification overhead that blunts the gains. We therefore present Know-The-Ropes (KtR), a framework that converts domain priors into an algorithmic blueprint hierarchy: tasks are recursively split into typed, controller-mediated subtasks, each solved zero-shot or with the lightest viable boost (chain-of-thought, micro-tune, self-check). Grounded in the No-Free-Lunch theorem, KtR trades the chase for a universal prompt for disciplined decomposition. On a Knapsack benchmark (3–8 items) three GPT-4o-mini agents raise accuracy from 3% zero-shot to 95% on size-5 instances after patching a single bottleneck agent. On the tougher Task-Assignment suite (6–15 jobs) a six-agent o3-mini blueprint hits 100% up to size 10 and $\geq$84% on sizes 13–15, versus $\leq$11% zero-shot. Algorithm-aware decomposition plus targeted augmentation thus turns modest models into reliable collaborators—no ever-larger monoliths required.

1 Introduction

Individual large language model (LLM) agents typically excel in the specific domain they are optimized for (Thirunavukarasu et al., 2023; Kasneci et al., 2023; Wu et al., 2023b), yet they struggle to achieve universal versatile (Zhang et al., 2024; Xu et al., 2024). A natural antidote is division of labor—splitting tasks into specialized agents that negotiate a joint answer—and early frameworks like Mixture-of-Agents indeed show that a well-orchestrated team can outperform its best member (Wang et al., 2024; Guo et al., 2024; Bo et al., 2024).

Yet large-scale audits have cooled that optimism. Firstly, each problem needs a well-designed prompt with significant manual effort. Even though some multi-agent frameworks seems to achieve satisfying results, when evaluation leakage and prompt over-fitting are removed, the headline boosts of naïve agent swarms slip to single-digit percentages—and can even turn negative once a task needs more than two or three coordination rounds (Pan et al., 2025; Zhu et al., 2025). Post-mortems trace the shortfall to a recurring pattern: ill-posed decompositions propagate ambiguity, loose role definitions create blind spots or duplication, verification layers either invoke brittle chain-of-thought heuristics or blow up the token budget, and every extra message compounds latency and cost almost quadratically with the number of rounds (Ye, 2025; Shu et al., 2024). In short, simply throwing more "brains" at a problem does not guarantee progress; sustainable gains demand disciplined, principled systems engineering—the very gap our work aims to close.

Know-The-Ropes (KtR) converts domain priors into an algorithmic hierarchy: recursively split a task until each leaf fits the base LLM's zero-shot reach or the lightest augmentation (chain-of-thought (Wei et al., 2022), small fine-tune, self-check loops, etc). Typed I/O contracts, enforced by a lightweight controller, isolate agents and prevent cross-talk, context bloat, or silent overwrites. Multi-agent system(MAS) construction thus resembles classic pipeline tuning—pinpoint the bottleneck, refine the split, and attach the cheapest fix that works.

Theory justifies the approach. The No-Free-Lunch theorem (Wolpert and Macready, 1997; Wolpert, 2021) dictates that no single agent or orchestration rule prevails across all problem distributions—there is no silver bullet. Robustness must therefore flow from exploiting domain structure, not from ever-larger prompt templates. KtR meets this mandate by re-using classical algorithms whose behavior is already well understood and op-

timized.

Our empirical preview includes the following:

• **Knapsack Problem (proof of concept with a lightweight base model).** On 3–8-item instances GPT-4o-mini scores only 60 % $\rightarrow$ 0 % zero-shot, and a blunt task-level fine-tune barely helps. Applying KtR yields a three-agent blueprint; fine-tuning just the "trimmer" on 1 200 examples lifts accuracy to 95 %–70 %, showing that KtR can turn a compact, low-capacity model into a high-performing multi-agent system.

• **Task-Assignment Problem (proof of scalability with a stronger base model).** With o3-mini, a six-agent KtR blueprint tackles sizes 6–15. Splitting one bottleneck agent into two finer leaves drives those leaves to 100 % and 97 % accuracy and raises overall system accuracy to $\geq$ 84 % across all sizes—demonstrating that KtR's gains grow with base-model capacity and that the knapsack results are not an isolated success.

Our contributions lie in two aspects.

• **KtR framework**. By formalizing blueprint hierarchies, we apply NFL-grounded algorithmic design that bypasses documented MAS pitfalls, streamlines the assembly pipeline, and relieves performance bottlenecks.

• **Empirical validation**. Across two canonical optimization problems, KtR transforms modest base models into systems that match—or exceed—their fine-tuned counterparts while using orders-of-magnitude less specialized data.

2 Related Work

Multi-Agent Systems (MAS) have been widely employed to enhance the capabilities of LLMs to tackle complex tasks (Qiu et al., 2024; Yan et al., 2024; Ma et al., 2024; Lin et al., 2024; Hua et al., 2023; Yu et al., 2024). This is because MAS typically distribute tasks across agents that collaborate to achieve a common goal, thereby improving both efficiency and adaptability. Recent frameworks like CAMEL (Li et al., 2023) enable role-based cooperative dialogues by assigning agents distinct personas, while AutoGen (Wu et al., 2023a) and MetaGPT (Hong et al., 2023) orchestrate multi-role agent teams through structured conversation loops and predefined workflows. In math optimization, OR-LLM-Agent can translate natural-language problem descriptions into formal Gurobi models—achieving an 85% correct-solution rate on real-world benchmarks (Zhang and Luo, 2025).

Despite this excitement, studies show that simply scaling up to LLM-based MAS often yields only marginal gains over single-agent baselines (Pan et al., 2025). LLM agents still struggle with context management and consistency, meaning that elaborate multi-agent prompts can fail to realize the intended collaboration (Bo et al., 2024). A recent systematic audit of popular MAS frameworks has identified 14 distinct failure modes (Cemri et al., 2025), which can be grouped into three categories, including flawed design (e.g., ambiguous role definition), inter-agent misalignment (e.g., communication failures), and quality control (e.g., no reliable check mechanism).

To address these challenges, researchers have proposed multiple strategies to make LLM-based MAS more reliable (Zhu et al., 2025; Tran et al., 2025). A key strategy is improving the agent interaction structure (Zhu et al., 2025). For example, the AgentDropout framework proposes a dynamic agent-pruning strategy, which seeks to discard less critical actors during training (Wang et al., 2025). Another effective strategy is incorporating feedback and verification loops (Hong et al., 2023). A recent study shows that frameworks with strong role specialization and iterative feedback mechanisms tend to outperform those without these features (Anonymous, 2025). In addition, systematic evaluations suggest that the communication topology matters: a well-designed protocol between agents can significantly improve collective performance on complex tasks (Zhu et al., 2025).

While existing multi-agent frameworks and strategies have demonstrated notable progress (Li et al., 2025; Hong et al., 2023; Zhu et al., 2025), they still face challenges in dynamic role reallocation and efficient inter-agent communication. These limitations become especially pronounced when addressing complex tasks, such as NP-hard optimization problems. To bridge this gap, our work proposes a heuristic strategy that embeds domain-specific rules and algorithms directly into agent coordination. This approach enables on-the-fly role adaptation and task decomposition, particularly in math optimization contexts where conventional MAS frameworks often struggle.

3 Methodology

3.1 Framework Design—Know the Ropes

We propose the heuristic framework "Know the Ropes." This framework offers a structured

methodology for designing specialized MAS leveraging LLMs. This heuristic focuses on translating known, effective procedures or algorithms into a coherent multi-agent architecture. As presented in Figure 1, the core idea is to decompose a complex overall task into its fundamental computational stages. Each stage is then mapped to a well-formulated sub-task, designed to be tractable for an individual agent. These specialized agents are subsequently orchestrated to mirror the data/control flow of the original procedure, which can effectively embed problem-solving logic into the multi-agent system. The following definitions formalize the components of this framework.

Definition 3.1. *A* **well-formulated task** *is a tuple $T = (I, O, R)$, consisting of the following:*

• Input domain I: an unambiguous description of all admissible inputs

• Output co-domain O an unambiguous description of all admissible outputs

• Requirement relation $R \subset I \times O$: a relation such that for each input $x \in I$ it defines explicitly the subset $R(x) \subset O$ as the set of outputs that are considered correct.

Definition 3.2. *A* **workflow blueprint** *$\mathcal{B} = (\mathcal{T}, \mathcal{P})$ consisting of the following.*

• A finite set of well-formulated tasks $\mathcal{T} = \{T_1, \cdots, T_n\}$.

• An orchestration protocol $\mathcal{P}$ that specifies:

– The control-flow graph that determines when each T_i is invoked.

– The data-dependency edges that map outputs of some tasks to inputs of others.

– Any global invariants, error-handling rules, or communication channels required to realize the end-to-end objective of $\mathcal{B}$.

Definition 3.3. *Given a workflow blueprint $\mathcal{B} = (\mathcal{T}, \mathcal{P})$, a* **decomposition** *selects a task $T \in \mathcal{T}$ and replace it with a sub-blueprint $\mathcal{B}_T = (\mathcal{T}_T, \mathcal{P}_T)$ such that the following hold.*

• Each task $T' \in \mathcal{T}_T$ is strictly simpler than T.

• The composite protocol $\mathcal{P}'$, obtained by embedding $\mathcal{P}_T$ in place of T inside $\mathcal{P}$, preserves all external interface of T.

The result of the decomposition is a new blueprint $\mathcal{S}' = (\mathcal{T}', \mathcal{P}')$, where $\mathcal{T}' = (\mathcal{T} \backslash \{T\}) \cup \mathcal{T}_T$. We record this process as

$$\mathcal{B} \overset{D}{\rightsquigarrow} \mathcal{B}'$$

Definition 3.4. *Let $\mathcal{M}$ be a set of LLM models. A well-formulated task T is said to be* **$\mathcal{M}$-tractable** *if a model inside $\mathcal{M}$ satisfies the requirement relation R_T with high, empirically verified accuracy, after optional augmentations (e.g., chain-of-thought prompting, tool calls, self-reflection loops, or fine-tuning).*

Definition 3.5. *Given a set of LLM models $\mathcal{M}$ and a blueprint $\mathcal{B}$, an* **$\mathcal{M}$-tractable hierarchy** *is a sequence of decompositions*

$$\mathcal{B} \overset{D_1}{\rightsquigarrow} \mathcal{B}_1 \overset{D_2}{\rightsquigarrow} \cdots \overset{D_n}{\rightsquigarrow} \mathcal{B}_n$$

such that each task in the terminal blueprint $\mathcal{B}_n$ is $\mathcal{M}$-tractable in the sense of Definition 3.4.

Definition 3.6. *Given a set of LLM models $\mathcal{M}$ and an $\mathcal{M}$-tractable blueprint $\mathcal{B} = (\mathcal{T}, \mathcal{P})$, a* **system instantiation** *is to instantiate $\mathcal{B}$ into a MAS in the following way.*

• We create one agent A_i per tractable task $T_i \in \mathcal{T}$, bundling any necessary augmentations with the agent.

• We implement the orchestration protocol $\mathcal{P}$ as message-passing or function calls among agents, preserving data-dependencies and control flow.

Algorithm 1 Know-The-Ropes (KtR) Pseudo-code

```
procedure KnowTheRopes(T, M)
    B ← CreateBlueprint({T}, trivial_protocol)
    while ∃ U ∈ B.tasks & ¬MTractable(U, M) do
        U* ← ChooseTaskToDecompose(U)
        B_sub ← DesignSubBlueprint(U*)
        B ← EmbedSubBlueprint(B, U*, B_sub)
    end while
    MAS ← InstantiateSystem()
    for all V ∈ B.tasks do
        aug ← SelectAugmentations(V, M)
        agent ← CreateAgent(V, M, aug)
        MAS.AddAgent(agent)
    end for
    ImplementProtocol(MAS, B.protocol)
    return MAS
end procedure
```

Our method. For a given (well-formulated) task T and a given set of LLM models $\mathcal{M}$ that will be used to solve the task, we perform the following. (See Algorithm 1 and Figure 1)

• Define the initial blueprint $\mathcal{B} = (\mathcal{T}, \mathcal{P})$, where $\mathcal{T} = \{T\}$ and $\mathcal{P}$ is trivial.

• Guided by domain heuristics, prior knowledge, and experiments on specific tasks, we construct an $\mathcal{M}$-tractable hierarchy

$$\mathcal{B} \overset{D_1}{\rightsquigarrow} \mathcal{B}_1 \overset{D_2}{\rightsquigarrow} \cdots \overset{D_n}{\rightsquigarrow} \mathcal{B}_n$$

• Materialize the terminal blueprint $\mathcal{B}_n$ as a MAS $\mathbf{MAS}(\mathcal{B}_n, \mathcal{M})$ that targets the initial task T.

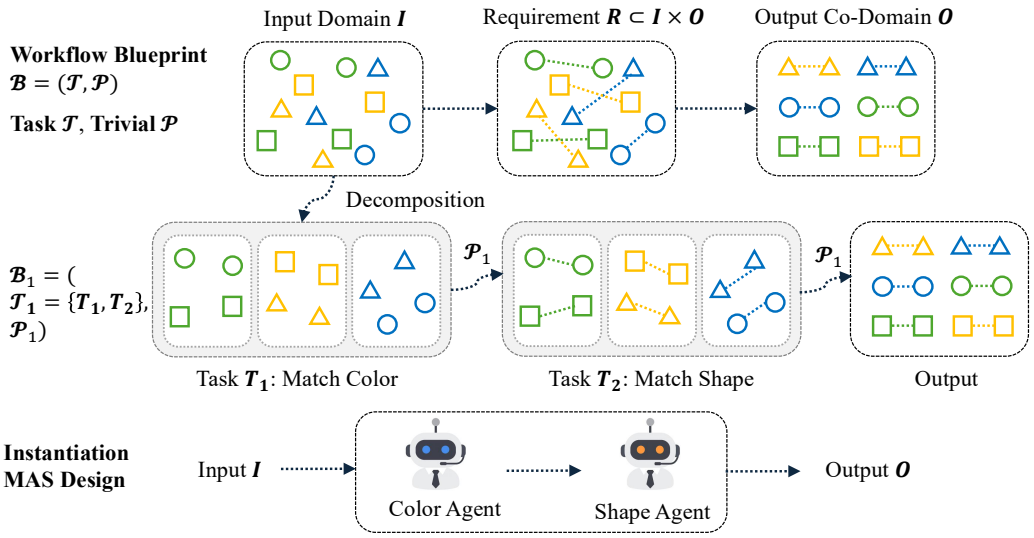

Figure 1: Illustration of the Know-The-Ropes (KtR) strategy: heuristic, prior-guided decomposition of a complex task into sub-tasks, each instantiated as a coordinated LLM agent within a multi-agent architecture.

This three-step procedure—atomic blueprint, tractable hierarchy construction, and system instantiation—provides a principled pathway from an arbitrarily complex task to a deployable multi-agent solution whose correctness hinges on model capabilities that have been explicitly validated. To demonstrate the practical application and efficacy of the "Know the Ropes" framework in creating such specialized MAS, we present two case studies. In each case, a well-understood algorithmic solution to a complex problem is decomposed into an M-tractable hierarchy and instantiated as a MAS.

4 Experiment Design

4.1 Proof-of-Concept: 0/1 Knapsack Problem (KSP)

To furnish a clear proof-of-concept for KtR, we start with the classical NP-hard Knapsack Problem (KSP)—a staple in resource allocation, logistics, and investment planning. By deliberately using the lightweight, general-purpose GPT-4o-mini as every agent's backbone, we establish a modest baseline that lets us highlight how KtR's multi-agent choreography amplifies a small model's capability well beyond its solo limits.

Below we only present a mathematical formulation of the problem, while a more detailed explanation of the problem as well as the algorithmic solution can be found in Appendix B.

4.1.1 Problem Formulation

For a Knapsack problem of size N, its input involves a weight vector $\vec{w} = (w_1, \cdots, w_N)$, a value vector $\vec{v} = (v_1, \cdots, v_N)$, and a capacity W. To formulate the goal, we introduce the state vector $\vec{x} = (x_1, \cdots, x_N) \in \{0,1\}^N$, i.e., all its entries take value in $\{0, 1\}$. Then the objective of the problem is to find the following value

$$Z = \max_{\substack{\vec{x} \in \{0,1\}^N \\ \vec{x} \cdot \vec{w} \leq W}} \vec{x} \cdot \vec{v}.$$

4.1.2 KtR Multi-Agent Design

Following KtR heuristic, the iterative dynamic programming solution for the KSP as in Appendix B.2 is decomposed into tasks for three specialized agents, as presented below. Prompts designed each individual agent are attached in Appendix D.

System Controller: Controller initialize a set of feasible states $S_0 = \{(0, 0)\}$ and controls a look on k from 1 to N. For each k, it sends S_{k-1} and (w_k, v_k) to Worker Agent, and then send the result plus W to the Trimmer Agent. The Controller then take the union of the output of Trimmer Agent and S_{k-1} to obtain S_k. Once all items are processed, the Controller invokes the Reporter Agent for final result.

Worker Agent: This agent is responsible for the iterative state expansion by the following formula:

$$S_{add} = \{(w + w_k, v + v_k) \text{ for all } (w, v) \in S_{k-1}\}.$$

Trimmer Agent: This agent performs the trimming task according to the following formula:

$$S_{trimmed} = \{(w, v) \in S_{add} \mid w \leq W\}.$$

Reporter Agent: This agent executes the solution report. It find the element with maximal value within the final state set S_N.

4.2 Proof of scalability—Task Assignment Problem (TAP)

Building on the previous section—where KtR already stretched the capabilities of the compact GPT-4o-mini on the Knapsack baseline—we now test KtR's scalability. We upgrade the backbone to the larger o3-mini and tackle the more demanding Task-Assignment Problem (TAP), demonstrating that the framework's performance rises in lockstep with the underlying model's capacity.

Again, below we only present a mathematical formulation and details are referred to Appendix B.

4.2.1 Problem Formulation

For a Task assignment problem of size N, its input involves an $N \times N$ cost matrix $C = (C_{ij})_{N\times N}$. We use $\mathfrak{S}_N$ to denote the set of permutations of n elements, or equivalently, the set of bijections from the set $\{1, 2, \cdots, N\}$ to itself. The objective of the problem is to find the following value:

$$Z = \max_{\sigma\in\mathfrak{S}_N} \left(\sum_{i=1}^{N} C_{i\sigma(i)} \right).$$

4.2.2 KtR Multi-Agent Design

Algorithm from Appendix B.4 now maps to a MAS under our KtR methodology. As explained in Section 5.2.2, based on test results of the agentic tasks and heuristic argument, we further decompose the tasks from the original system design to further improve the performance. The final system design contains six agents described below. Let N be the size of the problem and C be the original cost matrix.

Row Reducer: This agent reduces rows of the matrix C to obtain C'.

Column Reducer: This agent further reduces the columns of C' to obtain C''.

Matcher: This agent finds a maximal collection of zeroes in the reduced matrix C'', such that no two zeroes share same row or column. Let L be the number of zeroes in the maximal collection.

Painter: When $L < N$, with input from Mather, Painter is prompted for find a minimal collection of rows and columns covering all zeroes.

Normalizer: Receiving input from Painter, Normalizer creates more zeroes outside of the selected rows and columns to obtain an updated matrix C'''.

Reporter: When $L = N$, Report sums up values of entries of the original cost matrix C corresponding to the maximal collection of zeroes found by Matcher, and report this sum as the final solution to the problem.

The **System Controller** arranges task for Row Reducer and Column Reducer linearly, then controls a loop: Matcher first find a set of zeroes and the Controller checks if the number of zeroes L equals the problem size N, the size of the problem. If so, the loop is broken and the positions of zeroes is sent to the Reporter for final output. Otherwise, Painters are called in to find a minimal collections of lines to cover the zeroes and Normalizer follows to create more zeroes. Then the Controller iterates the loop and send the updated matrix to Matcher.

5 Experiment Result

Our experimental protocol unfolds in two stages. First, we run a uniform benchmark across a suite of baseline models—including several GPT and Llama variants—to fix a reference point for each task. The second stage then splits by objective: For KSP we deliver a proof of concept, while for TAP we provide a proof of scalability. For ground truth, we use python code to randomly generate problems, and then use the Google OR-Tools (Perron and Furnon, 2022) as in Appendix C to generate solutions to compare with.

5.1 Experiment Result for KSP

5.1.1 Baseline LLM Performance

Figure 2 shows the baseline LLM performance across multiple difficulty levels. The accuracy across difficulty levels (from 3 to 8 items) in the KSP scenario reveals substantial performance variation among the tested LLMs. Among those, the GPT-o3-mini, as a reasoning model, consistently demonstrates superior accuracy. Model GPT-4.1 outperforming its smaller counterparts, namely GPT-4.1-mini and its primer GPT-4o-mini. Other LLMs, including Claude-haiku, Llama, and Qwen series also show performance degradation, with higher variability particularly evident at greater difficulty levels. Meanwhile, the performance of final KtR multi-agent system is also drawn in Figure 2. The comparison shows that KtR substantially boosts performance, validating its effectiveness.

5.1.2 Multi-Agent Performance

Based on Figure 2, GPT-4o-mini exhibits a pronounced performance decline beginning at in-

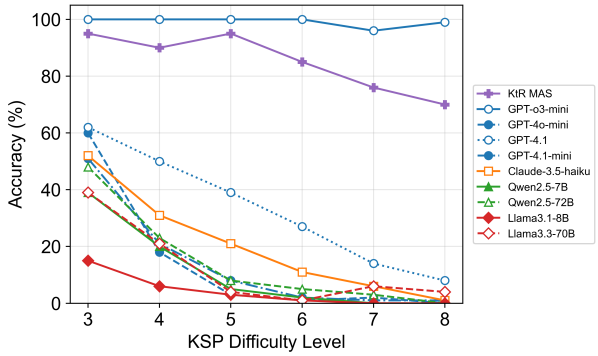

Figure 2: KSP baseline performance from single LLMs as well as the KtR mulit-agent system.

stances of 4 items, underscoring its limited scalability to more complex scenarios; thus, we select it as the baseline model for our KtR framework design. Figure 3 further illustrates the resulting multi-agent system along with the experimental outcomes derived from our proposed strategy.

Single LLM performance. We establish two baseline performances for GPT-4o-mini acting as a single agent to solve the KSP. First, the zero-shot GPT-4o-mini is directly prompted with KSP instances. As Figure 3B s shows, its accuracy decreases from 60% for 3 items to 0% (8 items). Second, we evaluate a fine-tuned GPT-4o-mini (standalone). Figure 3C indicates that fine-tuning offers some improvement over the zero-shot, but still as low as 3% for 8-item KSP.

Standard MAS performance. Following our KtR heuristic, we map the algorithm for KSP into a MAS design, illustrated in Figure 3F. Initially, each agent is driven by the standard, non-fine-tuned GPT-4o-mini. The performance of this standard MAS is presented in Figure 3F. Its performance descreases from 18% for 3 items to 4% for 8 items. This initial result implies that without augmenting the agents' abilities, the MAS does not effectively handle the task.

We profile each agent in isolation (Figure 3E) and uncover a single choke point: Trimmer. Its accuracy collapses as the feasible-state set S_k (cf. Section B.2) grows—54 % for 1–8 states, 24 % for 9–16, 7 % for 17–24, and just 5 % for 25–32. Because the algorithm loops once per state, even small per-iteration errors compound, and this cascading inaccuracy ultimately sinks the entire run.

Augmented MAS performance. To eliminate the bottleneck, we fine-tune the Trimmer's GPT-4o-mini backbone (Figure 3G, highlighted as 'Augmented Trimmer'). Accuracy leapt to 95 % for 1–8 feasible states, 89 % for 9–16, 81 % for 17–24, and 67 % for 25–32 (Figure 3H). Adding a lightweight self-check—prompting the model to audit its own answer—preserved or marginally improved these gains. Replacing the bottleneck with the fine-tuned Trimmer lifts end-to-end KSP accuracy to near-saturation across sizes (Figure 3I): 95 % for 3-item instances, 90 % for 4, 95 % for 5, 85 % for 6, 76 % for 7, and 70 % for 8. A single targeted upgrade thus turns KtR into a consistently high-performing solver as the problem scales.

5.2 Experiment result on TAP

5.2.1 Baseline LLM Performance

Figure 4 illustrates the baseline performance of multiple LLMs on the TAP task across multiple difficulty levels (from 3 to 8 tasks). The results reveal marked differences in model capabilities. The only reasoning model, GPT-4o-mini, consistently outperforms all others, exhibiting strong accuracy at lower difficulty levels, though its performance declines as task complexity increases. In contrast, GPT-4.1 demonstrates moderate but stable accuracy across all difficulty levels, surpassing its mini-sized counterparts. Other models, including Claude-3.5-Haiku, Qwen2.5, and Llma-3 variants, show intermediate performance with variability.

We observe that single-agent models (e.g., GPT-3-mini, GPT-4-mini, GPT-4.1) drop to 30-50% accuracy at TAP levels 7-8, while KtR MAS maintains steady performance near 100%, even surpassing reasoning models, demonstrating its exceptional robustness and generalization capabilities.

5.2.2 Multi-Agent Performance

Based on Figure 4, GPT-o3-mini consistently outperforms other LLMs across all evaluated tasks, making it our choice for subsequent experiments. Our goal is to assess the scalability of our proposed strategy and investigate how its performance evolves as task difficulty increases. Figure 5 illustrates the MAS design and corresponding experimental outcomes obtained using our heuristic-based approach.

Single LLM performance. Again, we evaluate the baseline performance of using o3-mini as a single agent. The o3-mini model achieves a relatively high performance (83%) but decays quickly as in Figure 5B: 37% on problems of size 9, 21% on problems of size 12 and finally is reduced to 3% for problems of size 15.

Agent performance and further decomposition. Guided by the Hungarian algorithm (Kuhn,

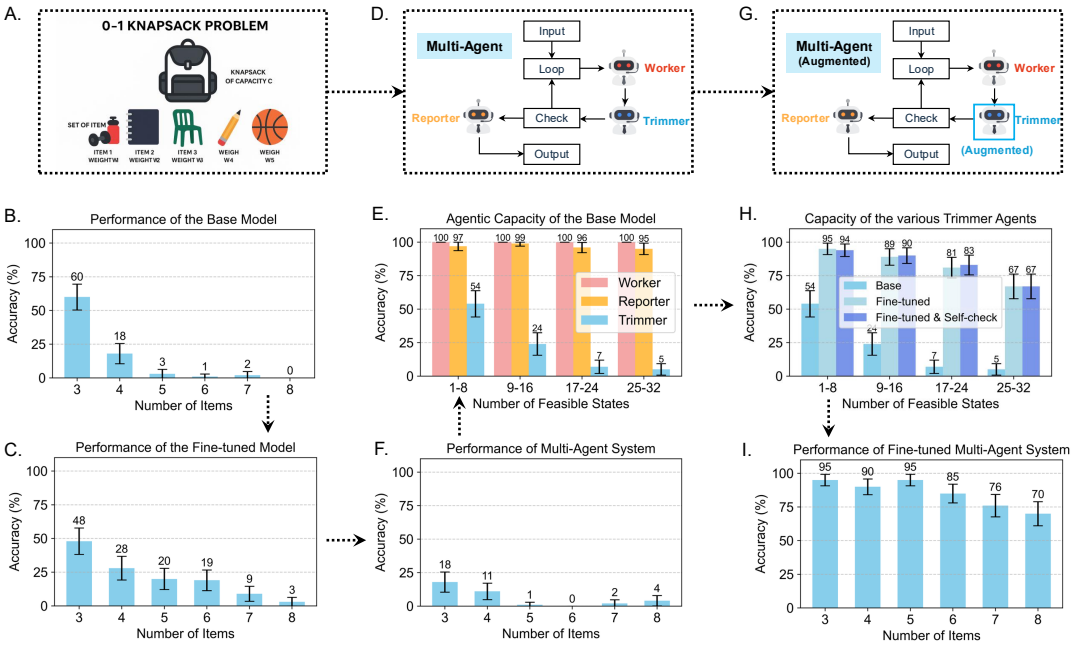

Figure 3: KSP evaluation of the KtR stategy. **B**: Zero-shot accuracy of the baseline model. **C**: Zero-shot accuracy after a light, task-specific fine-tune of the same model. **D** & **G**: Blueprints of the MAS without (**D**) and with (**G**) augmentations. **E**: Per-agent accuracies before augmentation, revealing the system's bottleneck. **H**: Boost delivered by two targeted augmentations—task-level fine-tuning and self-check prompting—applied to the bottleneck agent. **F** & **I**: Corresponding test accuracies for the two blueprints.

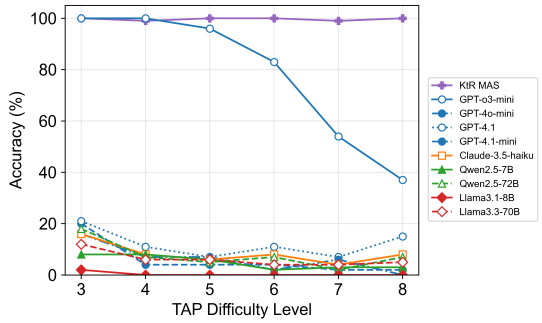

Figure 4: TAP baseline performance from single LLMs as well as the KtR mulit-agent system.

1955), our first KtR blueprint mapped each step to a single agent; Step 3 from Appendix B.4 relied on a lone Cover Seeker rather than the later "Matcher + Painter" pair. This baseline already scored 98 % (size 6), 88 % (size 9), 78 % (size 12), and 56 % (size 15), validating the approach.

We then stress-tested each agent on two bands, with matrix sizes 6-10 and 11-15, to pinpoint weaknesses. One-shot agents were flawless: Row Reducer and Reporter reached 100 % on both bands, and Column Reducer hit 100 % / 92 %. Normalizer held 99 % / 94 %, but Cover Seeker fell to 97 % / 84 %. Because Zero Seeker operates inside the main loop, its errors accumulate, making it the clear bottleneck for larger TAP instances.

We then perform a further decomposition of Step 3 in Section B.2 in a two-step process: Step 3.1. Finding a **maximal** collection of zero-entries, such that no two share a same row or column; Step 3.2. Finding a **minimal** collection of rows and columns covering all zero-entries.

We believe this decomposition is helpful due to the following reasons. First, by a mathematical argument, the size of collections from sub-tasks 3.1 and 3.2 match. Second, a heuristic argument indicating that knowing the maximal collection of zeroes simplifies the task to find minimal collection of rows and columns. Last, the original Step 5 can then simply use the positions of the zeroes from Step 3.1, once optimal check passes. Note, this also explains why we prefer a further decomposition rather than fine-tuning the original agent, as a further decomposition improves the system flow as well.

Empirical pays off, as shown in Figure 5I, Matcher reaches 100 % accuracy on both difficulty bands, while Painter climbs to 98 % and 97 %—a sharp jump from the original Cover Seeker's 97 % and 84 %.

Final MAS performance. Leveraging the refined decomposition, we deploy a six-agent system (Figure 5H) that solves size 6–10 instances almost flawlessly: almost 100 % accuracy versus 83 – 27 % for o3-mini zero-shot. It sustains high perfor-

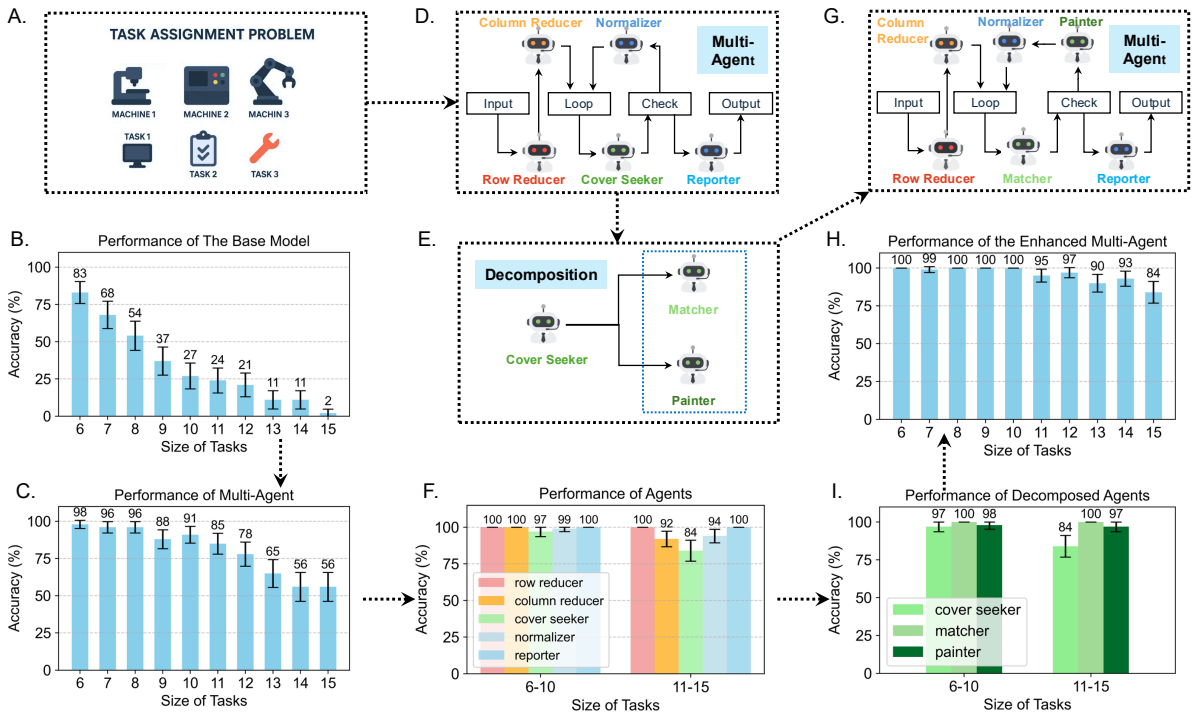

Figure 5: TAP evaluation of the KtR strategy. **B**: Zero-shot accuracy of the baseline model. **D**: Initial blueprint derived from the Hungarian algorithm; its end-to-end accuracy is shown in **C**. **F**: Per-agent accuracies within this blueprint, prompting the finer decomposition outlined in **E**. **I**: Side-by-side comparison of per-agent accuracies before and after decomposition. **G**: Final, decomposed blueprint, whose overall accuracy appears in **H**.

mance on size-11–15 tasks (95 %, 97 %, 90 %, 93 %, 84 %); even the dip at size 15 far exceeds the 3 % zero-shot baseline, highlighting the substantial capacity gain of our multi-agent architecture.

6 Discussion

Across two case studies, we show that disciplined decomposition combined with diagnosis-driven augmentation can raise low-capacity language models into dependable problem-solvers—and the gains grow further as the underlying model improves.

Proof-of-Concept on KSP. Baseline GPT-4o-mini accuracy collapsed as instance size increased, and a naïve three-agent blueprint offered little relief. Agent-level profiling singled out Trimmer as the lone bottleneck. Fine-tuning that agent alone—1 200 step-by-step examples, no changes elsewhere—lifted end-to-end accuracy from ≤ 18 % to ≥ 70 % across sizes 3–8, peaking at 95 % on size 3. These results validate KtR's "identify $\rightarrow$ isolate $\rightarrow$ augment" cycle: a targeted upgrade can rescue a small general-purpose model when both single-agent and untargeted multi-agent baselines fail.

Proof-of-Scalability on TAP. Repeating the procedure with the stronger o3-mini model, we began with a five-agent design. Direct testing pinpointed a composite planning task that capped overall performance. KtR prescribes recursive refinement: we split that task into two simpler, typed sub-tasks, each amenable to the base model. Accuracy on the difficult sizes (11-15) rose from as low as 11 % single-agent to ≥ 84 % multi-agent, while sizes 6-10 reached 100 %. Thus, as model capacity scales, KtR continues to amplify it rather than saturate.

Beyond these demonstrations, we would like to also outline several research threads that would deepen the approach in the future:

- **Model Portfolio Allocation.** Many leaves sit well below the flagship model's frontier; mixing lighter or domain-tuned LLMs would slash cost while keeping accuracy, letting the controller pick "just-enough" capacity per task.
- **Complexity–Capacity Estimation.** Replace rule-of-thumb splits with a principled score that (i) quantifies task difficulty and (ii) predicts an LLM's post-augmentation capacity, enabling data-driven decompositions.
- **End-to-End Automation**. With the above metrics, the ultimate goal is to design a system that automates the whole KtR methodology: decomposing the tasks, evaluate capacities, and assemble the multi-agent system for solution.

7 Limitations

Despite demonstrating sizable performance gains, our study has several limitations that should guide future work.

Narrow task scope. We evaluate KtR on two canonical optimization problems (Knapsack and Task-Assignment). While chosen to illustrate proof-of-concept and proof-of-scalability, these tasks have well-structured objective functions and small action spaces; generalizing to open-domain reasoning or multi-modal settings remains untested.

Synthetic data & idealized inputs. All problem instances are randomly generated and fully specified. Real-world inputs (noisy, partially observed, or adversarial) could degrade both decomposition quality and agent reliability.

Cost and latency trade-offs. Although the per-agent inference cost is trending downward, we do not quantify absolute wall-clock latency, energy consumption, or controller overhead for large agent swarms.

Bottleneck identification heuristic. We locate bottleneck subtasks via held-out accuracy screens; this assumes the availability of inexpensive ground-truth labels. Automated bottleneck detection without labels is an open problem.

8 Ethical considerations

Amplifying decision-making power by scaling agent crowds may exacerbate existing biases present in the base models; we have not run a bias or fairness audit. Addressing these limitations—particularly expanding to less structured domains and benchmarking real-world resource usage—will be essential to establish the broader utility and safety of the KtR framework.

A Appendix: Weighted No Free Lunch Theorem

In this section, we present a weighted version of the No-Free-Lunch theorem. As the motivation, current approaches to MAS design can often result in overly general solutions that may exhibit suboptimal performance on specific and complex tasks. This sub-optimality arises partially from a lack of domain-specific inductive bias. To formalize this, we present a weighted variant of the No Free Lunch (NFL) theorem. The following demonstration, leveraging a weighted variant of the No Free Lunch theorem, quantitatively illustrates how inductive bias tailored to the target domain enhances performance.

That is, we present a formal proof showing that, under a non-uniform prior concentrated on a problem-specific subset of functions, a specialized learning algorithm achieves strictly lower expected risk than a general-purpose algorithm.

Note the No-Free-Lunch theorem has been known to research community for more than two decades. Here what we present is a modification of the standard statement to better fit for our discussion on the MAS. As we don't find in literature the precise version of the NFL theorem as we stated below, we also present a proof for self-containedness. We do not claim any originality of the theorem and the proof.

Theorem A.1 (Weighted NFL)**.** *Let X be a finite input domain, Y a finite label set, and $\mathcal{F} = Y^X$ the set of all functions $f\colon X \to Y$. Consider*

- *a general algorithm A_0 with constant expected loss ε_0 on every $f \in \mathcal{F}$,*
- *a specialized algorithm A' satisfying*

$$L(h_{A'}, f) \le \begin{cases} \varepsilon_1, & f \in \mathcal{F}', \\ \varepsilon_2, & f \notin \mathcal{F}', \end{cases}$$

  where $\varepsilon_1 < \varepsilon_0 < \varepsilon_2$, and

- *a prior P with $P(f \in \mathcal{F}') = p$ and $P(f \notin \mathcal{F}') = 1 - p$.*

If

$$p > \frac{\varepsilon_0 - \varepsilon_2}{\varepsilon_1 - \varepsilon_2},$$

then the expected risk of A' is strictly lower than that of A_0, i.e.

$$R(A') < R(A_0).$$

Proof. By definition,

$$R(A_0) = \mathbb{E}_{f\sim P}[L(h_{A_0}, f)] = \varepsilon_0$$

and

$$R(A') = \mathbb{E}_{f\sim P}[L(h_{A'}, f)] = p\,\varepsilon_1 + (1-p)\,\varepsilon_2.$$

Hence

$$\begin{aligned} R(A') < R(A_0) &\Longleftrightarrow p\,\varepsilon_1 + (1-p)\,\varepsilon_2 < \varepsilon_0 \\ &\Longleftrightarrow p(\varepsilon_1 - \varepsilon_2) > \varepsilon_0 - \varepsilon_2, \end{aligned}$$

which rearranges to

$$p > \frac{\varepsilon_0 - \varepsilon_2}{\varepsilon_1 - \varepsilon_2}.$$

This completes the proof. □

B Appendix: KSP and TAP description

In this appendix, we provide details about the KSP and TAP, including their problem description and algorithm based on which we design our MAS.

B.1 KSP Problem Formulation

The usual input of KSP involves a set of N items, whose items are characterized by pairs (w_i, v_i) of weights w_i and values v_i, as well as a capacity value W. The goal of KSP is to find a subset of items such that the total weight does not exceed the given capacity while the total value is maximized. Mathematically, we record information of items by two vectors, both of dimension N: a weight vector $\vec{w} = (w_1, \cdots, w_N)$ and a value vector $\vec{v} = (v_1, \cdots, v_N)$. We also introduce the set of state-vectors $\{0,1\}^N$, whose elements are vectors $\vec{x} = (x_1, \cdots, x_N)$ where entries x_i takes values between 0 and 1, indicating whether an item is chosen in a subset or not:

$$x_i = \begin{cases} 1 & \text{item } i \text{ is chosen} \\ 0 & \text{item } i \text{ is excluded} \end{cases}$$

Thus state vectors controls which items is in the chosen subset, and the inner product of $\vec{x}$ with $\vec{w}$ and $\vec{v}$ then compute the total weight and total value for the given subset, respectively.

Given a weight vectors $\vec{w}$, a value vector $\vec{v}$, and the capacity constraint W, the objective of the Knapsack problem then can be formulated as finding the following (optimal) value:

$$Z = \max_{\substack{\vec{x}\in\{0,1\}^N \\ \vec{x}\cdot\vec{w}\leq W}} \vec{x}\cdot\vec{v}. \tag{1}$$

Here the maximal value is taken over all state vectors (or equivalently, all subsets of items) satisfying the constraint that the total weight $\vec{x}\cdot\vec{w}$ not exceeding the capacity W.

This version of the problem, where each item can either be fully included or not at all, is commonly known as the 0/1 KSP.

B.2 KSP Problem Solution

A classic approach to the Knapsack Problem iteratively enumerates all feasible states—a dynamic-programming strategy first introduced by Bellman (Bellman, 1957). A feasible state can be defined by a pair $(current\_weight, current\_value)$ representing the accumulated weight and value of a set of items selected so far, such that $current\_weight \leq W$. We can describe the algorithm in the form of mathematical induction. We start with the initial set of feasible states $S_0 = \{(0,0)\}$, representing an empty set of chosen items. We then add items in to form a set S_k from S_{k-1} inductively, with capacity being aware: for each k, assuming that S_{k-1} has been constructed, then we add the pair (w_k, v_k) to all items in S_{k-1} to form a new set S_{add}:

$$S_{add} = \{(w + w_k, v + v_k) \text{ for all } (w,v) \in S_{k-1}\}.$$

Then, we trim the set according to the capacity:

$$S_{trimmed} = \{(w,v) \in S_{add} \mid w \leq W\}.$$

Note this also removes all repetitive states in the set. Finally we take the union of the two intermediate sets to create S_k:

$$S_k = S_{k-1} \cup S_{trimmed}.$$

The inductive step terminate when we have run through all items and obtaining the final set S_N, we pick the element in S_N with maximal value, as the solution to the KSP. Explicitly,

$$Z = \max_{(w,v)\in S_N} v$$

B.3 TAP Problem Formulation

TAP seeks to optimally assign a set of N resources (agents or workers) to N tasks, where each potential assignment incurs a specific cost. With the constraint that each resource can only be assigned to one unique task, the objective of TAP is to find an assignment covering all tasks that minimizes the total cost. The resource-task specific cost is recorded in an $N \times N$ matrix C, where the entry C_{ij} represents the cost associated with assigning resource i to task j, for $i, j \in \{1, 2, \ldots, N\}$.

To formally define the problem, we introduce a set $\mathfrak{S}_n$ which can be described in either one of the following three equivalent ways:

• The group of automorphisms of the set $\underline{N} = \{1, 2, \cdots, N\}$.

• The set (or group) of bijections from the set $\underline{N}$ to itself.

• The set of all permutations involving N elements.

Note elements in $\mathfrak{S}_N$ convey the idea that each resource is assigned to a unique task.

Now, given the $N \times N$ cost matrix C, the objective of the TAP is to find the following (optimized) value

$$Z = \max_{\sigma \in \mathfrak{S}_N} \left(\sum_{i=1}^{N} C_{i\sigma(i)} \right).$$

Note when we treat $\sigma \in \mathfrak{S}_N$ as a (bijective) map from $\underline{N} = \{1, 2, \cdots, N\}$ to itself, the notation $C_{i\sigma(i)}$ represents the entry on the i-th row and $\sigma(i)$-th column of the cost matrix C.

B.4 TAP Problem Solution

The typical solution for TAP is using the Hungarian algorithm (Kuhn, 1955), which provides a polynomial-time method to find the objective value Z. We summarize the algorithm as follows.

Step 1. Row Reduction. For each row, we find the minimal element in the row and subtract it from all entries in the row, creating at least one zero on each row. Mathematically, starting from the original cost matrix $C_{N\times N}$, we create a new reduced matrix C' such that for $i, j \in \{1, 2, \cdots, N\}$, we have

$$C'_{ij} = C_{ij} - \min_{1 \le k \le N} C_{ik}$$

Step 2. Column Reduction. Similarly, we further reduce C' to C'' as follows: For each column, we find the minimal element in the column and subtract it from all entries in the column, guaranteeing at least one zero on each column. Mathematically, take

$$C''_{ij} = C'_{ij} - \min_{1 \le k \le N} C_{kj}$$

Step 3. Find covering lines. We then find a smallest collection of rows and columns to cover all zeroes. Here smallest is the sense of the number of elements in the collection (of rows and column), over all possible such collections. If the size of this minimal collection, denoted by L, coincides with N, the size of the problem, then we skip Step 4 to enter the final stage of the algorithm.

Step 4. Matrix Improvement. However, if $L < N$, we need an improvement for the matrix C'' before looping back to Step 3: given the minimal collection of rows and columns from Step 3, we find the minimal value of all entries that are not covered, and then subtract this minimal value from all uncovered entries of C'', and then add this minimal value to all entries of C'' that are covered twice, i.e., by both rows and columns. Let C''' be the resulting matrix.

Step 5. Assignment Identification. Once the condition $L = N$ is met, the final step is to identify the optimal assignment. This involves selecting a set of N independent zeros from the current matrix C''', such that no two selected zeros share the same row or column. Each selected zero at position (i, j) corresponds to assigning agent i to task j. The total cost of this optimal assignment is then calculated by summing the costs from the original cost matrix C corresponding to these selected zero positions.

A non-trivial fact guaranteed by the Hungarian algorithm is that, in Step 5 the collection of zeroes might not be unique, while different collections are deemed to result in the same total summation of corresponding entries in the original cost matrix C.

C Ground-truth Data Preparation

We utilize Google OR-Tools (Perron and Furnon, 2022) to generate optimal solutions—serving as ground-truth datasets—for both problem scenarios. OR-Tools is a widely adopted open-source software suite developed by Google for solving combinatorial optimization problems. It is renowned for its efficiency and reliability in addressing NP-hard challenges through advanced optimization algorithms. The suite is distributed under the permissive Apache License 2.0, allowing unrestricted use, modification, and distribution (Perron and Furnon, 2022).

For KSP, we generate random instances by assigning weights and values to items along with a maximum capacity constraint. Optimal solutions are then computed using OR-Tools' dynamic programming approach. For TAP, we similarly generate random cost matrices that represent the cost of assigning workers to tasks. Optimal assignments are obtained using the Hungarian algorithm as implemented in OR-Tools, which efficiently minimizes the total assignment cost.

D Appendix: Prompt gallery

Note that all prompts we presented in the following, except for the self-check prompt for Trimmer Agent in KSP problem, are the system prompt for agents. The user prompt will only contain the precise problem to be handled by the agent in the form specified by the prompt.

D.1 KSP prompts

D.1.1 Prompt for zero shot

You are an expert in the field of Knapsack Problem.

You are given a Knapsack Problem in the json format, of the following form:
{
"id" : str,
"items" : list of pairs of integers,
"capacity" : int
}

Each pair in the list is a pair of integers of the form [weight, value], i.e., the first entry is the weight and the second entry is the value.

Your task is to solve the Knapsack Problem and provide the optimal solution. That is, you need to find a subset of the pairs that maximizes the total value, subject to the constraint that the total weight of the subset is less than or equal to the capacity.

Please think step by step when solving the problem.

You need to return the optimal solution in the following json format:
{
"max_value" : int,
}
Please only return the json format, nothing else.

D.1.2 Prompt for Worker Agent

You are a key member of a multi−agent team collaboratively solving the Knapsack Problem. Your specific role is the Worker, responsible for performing mathematical computations for the team.

You will receive input in the following JSON format:
{"c_list": [[int, int], ...], "s_item": [int, int]}
Each pair within 'c_list' contains two integers.

Your task is to:
− Add 's_item' to each pair in 'c_list' entry−wise. For instance, if a pair in 'c_list' is '[2, 5]' and 's_item' is '[3, 4]', the result should be '[2+3, 5+4] = [5, 9]'.

To ensure accuracy:
− Proceed systematically, applying step−by−step reasoning.
− Carefully perform each addition individually for all pairs provided in the list.

Your response must strictly follow this JSON format:
{"n_list": [[int, int], ...]}

Return only the specified JSON object without any additional commentary or text.

D.1.3 Prompt for Trimmer Agent

You are a key member of a multi−agent team collaboratively solving the Knapsack Problem. Your specific role is the Trimmer, responsible for trimming the list based on the given capacity constraint.

You will receive input in the following JSON format:
{"n_list": [[int, int], ...], "capacity": int}

Each pair within 'n_list' contains two integers: the first integer represents the weight, and the second integer represents the value.

Your task is to:
− Carefully analyze each pair in the provided list.
− Remove all pairs whose weight (the first integer) strictly exceeds the specified capacity.
− If identical pairs appear multiple times, retain only one instance of each.

To ensure accuracy:
− Proceed systematically, applying step−by−step reasoning.
− Verify each pair carefully against the capacity constraint.

Your response must strictly follow this JSON format:
{"t_list": [[int, int], ...]}

Return only the specified JSON object without any additional commentary or text.

D.1.4 Prompt for Reporter Agent

You are a key member of a multi−agent team collaboratively solving the Knapsack Problem. Your specific role is the Reporter, responsible for determining and clearly reporting the final answer based on the provided information.

You will receive input in the following JSON format:
{"c_list": [[int, int], ...]}
Each pair within 'c_list' contains two integers: the first integer represents the weight, and the second integer represents the value.

Your task is to carefully analyze this list, identify the pair with the maximal value (the second integer in each pair), and report only that maximal value. If the list is empty, then report the maximal value as 0.

To ensure accuracy:
– Proceed systematically, applying step–by–step reasoning.
– Carefully examine every pair in the provided list.

Your response must strictly follow this JSON format:
{"max_value": int}

Return only the JSON object as specified above, without any additional commentary or text.

D.1.5 Self-check prompt for Trimmer Agent

To better fulfill your task, please conduct a double check on the result you just provided. If your answer is already correct, please confirm by copying the last output.

When double check, please pay attention to the following typical types of mistakes:

In particular, please check if you made any typical mistakes as listed below:
1. If you added in a pair that is not in the original n_list.
2. If there is still a pair in the t_list that still exceeds the capacity.
3. If there is a pair in n_list that does not exceed the capacity but is not in the t_list.

If you found any errors, please create a corrected answer.

In either case, please follow the format requirement of the output.

D.2 TAP prompts

D.2.1 Prompt for zero shot

You are an expert in solving the Assignment Problem. In the assignment problem, there are n workers and n jobs. Each worker has a cost of assigning to each job. Each worker can only be assigned to one job. Your task is to find the optimal assignment of workers to jobs that minimizes the total cost.

You are given the problem in the following json format:

```
{
  "id" : str,
  "cost_matrix" : list of lists of integers
}
```

The cost matrix is a square matrix of size n x n, where n is the number of workers and jobs, in the form of a nested list [[int, int, ...], [int, int, ...], ...]. The (i, j)th entry of the matrix represents the cost of assigning the ith worker to the jth job.

Your task is to find the optimal assignment of workers to jobs that minimizes the total cost.

Please think step by step when solving the problem.

You need to return the optimal assignment in the following json format:

```
{
  "optimal_cost" : int
}
```

Please only return the json format, nothing else.

D.2.2 Prompt for Row Reducer Agent

You are given a matrix in the following json format:

```
{
  "matrix" : list of lists of integers
}
```

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].

Your task is to reduce the matrix by subtracting the minimum value of each row from all the elements in that row.

Please think step by step when solving the problem:
Step 0: Work on one row at a time.
Step 1: Find the minimum value of the row.
Step 2: Subtract the minimum value of the row from all the elements in that row.
Step 3: Return the reduced matrix in the following json format:

{"reduced_matrix" : list of lists of integers}

Please only return the json format, nothing else.

D.2.3 Prompt for Column Reducer Agent

You are given a matrix in the following json format:

{
"matrix" : list of lists of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].

Your task is to reduce the matrix by subtracting the minimum value of each column from all the elements in that column.

Please think step by step when solving the problem:
Step 0: Work on one column at a time.
Step 1: Find the minimum value of the column.
Step 2: Subtract the minimum value of the column from all the elements in that column.
Step 3: Return the reduced matrix in the following json format:

{"reduced_matrix" : list of lists of integers}

Please only return the json format, nothing else.

D.2.4 Prompt for Cover Seeker Agent

You are given a problem in the following json format:

{
"matrix" : list of lists of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].

Your task is to find a smallest collection of rows and columns of the matrix, such that any zeroes in the matrix is contained in a chosen row or column. Small means the sum of the sizes of the row and column collections is the smallest possible.

Please think step by step when solving the problem, and return your response in the following json format:

{"collum_collection" : [int, int, ...], "
row_collection" : [int, int, ...]}

The integers in the collum_collection and row_collection are the indices of the rows and columns that you choose.

Please only return the json format, nothing else.

D.2.5 Prompt for Matcher Agent

You are given a matrix in the following json format:

{
"matrix" : list of lists of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].

Your task is to find the largest collection of zeroes in the matrix, such that no two zeroes are in the same row or column.

Please think step by step when solving the problem, and return your response in the following json format:

{"largest_collection" : [[int, int], [int, int], ...]}

The list of pairs of integers is in the form of [[row_index, column_index], [row_index, column_index], ...].

Please only return the json format, nothing else.

D.2.6 Prompt for Painter Agent

You are given a problem in the following json format:

{
"matrix" : list of lists of integers
"collection" : list of lists of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].

The collection is in the form of a nested list [[int, int], [int, int], ...].

Your task is to find a smallest collection of rows and columns of the matrix, such that any zeroes in the matrix is contained in a chosen row or column. Small means the sum of the sizes of the row and column collections is the smallest possible.

To assist you, you are provided with a collection of zeroes in the input json format. The collection contains the positions of a maximal collection of zeroes in the matrix, such that no two zeroes are in the same row or column.

Please use this collection of zeroes to find the rows and columns as desired. More precisely, you should first choose one row or column for each zero in the collection, such that the chosen rows and columns cover as much of the zeroes in the matrix as possible. Then add in more rows or columns if needed.

Please think step by step when solving the problem, and return your response in the following json format:

{"collum_collection" : [int, int, ...], "
row_collection" : [int, int, ...]}

The integers in the collum_collection and row_collection are the indices of the rows and columns that you choose.

Please only return the json format, nothing else.

D.2.7 Prompt for Normalizer Agent

You are given a problem in the following json format:

{
"matrix" : list of lists of integers
"collumn_collection" : list of integers
"row_collection" : list of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].
The collumn_collection and row_collection are the indices of some selected rows and columns that covers all the zeroes in the matrix.

Your task is the following:
1. Find the minimal value in the matrix that is not covered by the selected rows and columns.
2. If this value is 0, return the original matrix.
3. If this value is not 0, subtract this value from all uncovered entries in the matrix.
4. For the entries that covered by both a selected row and a selected column, add this value to the entries.
5. For the entries that are covered by a selected row or column, but not both, do nothing.
6. Please return the updated matrix in the following json format:

{"normalized_matrix" : list of lists of integers}

Please only return the json format, nothing else.

D.2.8 Prompt for Reporter Agent

You are given a problem in the following json format:

{
"matrix" : list of lists of integers
"collection" : list of lists of integers
}

The matrix is in the form of a nested list [[int, int, ...], [int, int, ...], ...].
The collection contains a set of entries of the matrix in the form of [[row_index, column_index], [row_index, column_index], ...].

Your task is the following:
1. Sum up the values of all the entries in the collection.
2. Return the total value in the following json format:

{"total_value" : int}

Please only return the json format, nothing else.

References

Anonymous. 2025. Code in harmony: Evaluating multi-agent frameworks. In *Submitted to CS598 LLM Agent 2025 Workshop*. Under review.

Richard Bellman. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA.

Xiaohe Bo, Zeyu Zhang, Quanyu Dai, Xueyang Feng, Lei Wang, Rui Li, Xu Chen, and Ji-Rong Wen. 2024. Reflective multi-agent collaboration based on large language models. *Advances in Neural Information Processing Systems*, 37:138595–138631.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, and 1 others. 2025. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. 2024. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*.

Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, and 1 others. 2023. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 3(4):6.

Wenyue Hua, Lizhou Fan, Lingyao Li, Kai Mei, Jianchao Ji, Yingqiang Ge, Libby Hemphill, and Yongfeng Zhang. 2023. War and peace (waragent): Large language model-based multi-agent simulation of world wars. *arXiv preprint arXiv:2311.17227*.

Enkelejda Kasneci, Kathrin Seßler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, Georg Groh, Stephan Günnemann, Eyke Hüllermeier, and 1 others. 2023. Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences*, 103:102274.

Harold W. Kuhn. 1955. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Hang Li, Tianlong Xu, Ethan Chang, and Qingsong Wen. 2025. Knowledge tagging with large language model based multi-agent system. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 28775–28782.

Shuhang Lin, Wenyue Hua, Lingyao Li, Che-Jui Chang, Lizhou Fan, Jianchao Ji, Hang Hua, Mingyu Jin, Jiebo Luo, and Yongfeng Zhang. 2024. BattleAgent: Multi-modal dynamic emulation on historical battles to complement historical analysis. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 172–181, Miami, Florida, USA. Association for Computational Linguistics.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Liu Boyin, Xiaolin Ai, Yanyan Liang, and Min Chen. 2024. Coevolving with the other you: Fine-tuning llm with sequential cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 37:15497–15525.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, and 1 others. 2025. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

Laurent Perron and Vincent Furnon. 2022. Google OR-Tools. https://developers.google.com/optimization.

Jianing Qiu, Kyle Lam, Guohao Li, Amish Acharya, Tien Yin Wong, Ara Darzi, Wu Yuan, and Eric J Topol. 2024. Llm-based agentic systems in medicine and healthcare. *Nature Machine Intelligence*, 6(12):1418–1420.

Raphael Shu, Nilaksh Das, Michelle Yuan, Monica Sunkara, and Yi Zhang. 2024. Towards effective genai multi-agent collaboration: Design and evaluation for enterprise applications. *arXiv preprint arXiv:2412.05449*.

Arun James Thirunavukarasu, Darren Shu Jeng Ting, Kabilan Elangovan, Laura Gutierrez, Ting Fang Tan, and Daniel Shu Wei Ting. 2023. Large language models in medicine. *Nature medicine*, 29(8):1930–1940.

Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D Nguyen. 2025. Multi-agent collaboration mechanisms: A survey of llms. *arXiv preprint arXiv:2501.06322*.

Junlin Wang, Jue Wang, Ben Athiwaratkun, Ce Zhang, and James Zou. 2024. Mixture-of-agents enhances

large language model capabilities. *arXiv preprint arXiv:2406.04692*.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. 2025. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. *arXiv preprint arXiv:2503.18891*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

David H Wolpert. 2021. What is important about the no free lunch theorems? In *Black box optimization, machine learning, and no-free lunch theorems*, pages 373–388. Springer.

David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, and 1 others. 2023a. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023b. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. 2024. Hallucination is inevitable: An innate limitation of large language models. *arXiv preprint arXiv:2401.11817*.

Yuwei Yan, Qingbin Zeng, Zhiheng Zheng, Jingzhe Yuan, Jie Feng, Jun Zhang, Fengli Xu, and Yong Li. 2024. Opencity: A scalable platform to simulate urban activities with massive llm agents. *arXiv preprint arXiv:2410.21286*.

Ye Ye. 2025. Task memory engine (tme): Enhancing state awareness for multi-step llm agent tasks. *arXiv preprint arXiv:2504.08525*.

Huizi Yu, Jiayan Zhou, Lingyao Li, Shan Chen, Jack Gallifant, Anye Shi, Xiang Li, Wenyue Hua, Mingyu Jin, Guang Chen, and 1 others. 2024. Aipatient: Simulating patients with ehrs and llm powered agentic workflow. *arXiv preprint arXiv:2409.18924*.

Bowen Zhang and Pengcheng Luo. 2025. Or-llm-agent: Automating modeling and solving of operations research optimization problem with reasoning large language model. *arXiv preprint arXiv:2503.10009*.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and 1 others. 2025. Multiagentbench: Evaluating the collaboration and competition of llm agents. *arXiv preprint arXiv:2503.01935*.